Current Journal of Applied Science and Technology

25(2): 1-9, 2017; Article no.CJAST.38361
Previously known as British Journal of Applied Science & Technology
ISSN: 2231-0843, NLM ID: 101664541

Reducing the Cost of Exploring Neighborhood Areas in Dynamic Local Search for SAT

Abdelraouf Ishtaiwi[1*], Ghassan Issa[1] and Wael Hadi[1]

[1]Faculty of IT, School of CS, University of Petra, Amman, Jordan.

Authors' contributions

This work was carried out in collaboration between all authors. Author AI designed the study, performed the statistical analysis, wrote the protocol and wrote the first draft of the manuscript. Authors GI and WH managed the analyses of the study. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/CJAST/2017/38361
Editor(s):
(1) Wei Wu, Professor, Department of Applied Mathematics, Dalian University of Technology, China.
Reviewers:
(1) Sanjib Kumar Datta, University of Kalyani, India.
(2) Utku Kose, Suleyman Demirel University, Turkey.
(3) K. Ashwini, Visvesvaraya Technological University, India.
(4) Sofiya Ostrovska, Atilim University, Turkey.
Complete Peer review History: http://www.sciencedomain.org/review-history/22549

Original Research Article

Received 23rd November 2017
Accepted 13th December 2017
Published 1st January 2018

ABSTRACT

Stochastic Local Search (SLS) algorithms are of great importance to many fields of Computer Sciences and Artificial Intelligence. This is due to their efficient performance when applied for solving randomly generated satisfiability problems (SAT). Our focus in the current work is on one of the SLS dynamic weighting approaches known as multi-level weight distribution (mulLWD). We experimentally investigated the performance and the weight behaviors of mulLWD. Based on our experiments, we observed that the 2nd level weights movements could lead to poor performance of mulLWD, especially when applied for solving large and harder SAT problems. Therefore, we developed a new heuristic that could reduce the cost of the 2nd level neighborhood exploitation known as partial multi-level weight distribution mulLWD+. Experimental results indicate that mulLWD+ heuristic has significantly better performance than mulLWD in a wide range of SAT problems.

Keywords: Artificial intelligence; Boolean satisfiability; search algorithms.

**Corresponding author: E-mail: aishtaiwi@uop.edu.jo;*

1. BACKGROUND

The Boolean satisfiability problems (also known as propositional satisfiability or SAT for short) form a substantial portion of the core of many computer science fields and artificial intelligence. This is due to the fact that, finding a satisfying assignment to a SAT problem is imperative and has very deep signification. In our research we consider formulas in Conjunctive Normal Form (CNF), where (CNF): $\mathcal{F} = \bigwedge_m \bigvee_n l_{mn}$ in which each l_{mn} is a literal (boolean variable or its negation), and each disjunct $\bigvee_n l_{mn}$ is a clause. The task of any given search algorithm is to find a complete assignment that satisfies $\mathcal{F}$. Due to the time limitation, this task is beyond the reach of systematic search algorithms except for limited small sized problems as SAT problems is NP complete. On the other hand, almost any simple stochastic local search (SLS) method could successfully solve a broad range of larger and more challenging problems [1].

Breakout heuristic [2] was introduced in 1993 as the first sls weighting algorithm and since then many tries have been made to enhance the performance of SLS weighting algorithms [3,4]. However, the performance of these algorithms remained weak when compared to non-weighting SLS techniques. This remained the case until the occurrence of Dynamic Lo- cal Search algorithms (DLS) such as DLM [5], SAPS [6], PAWS [7], DDFW [8] and more recently BalancedZ [9], probSAT [10] and DCCAlm [11].

Escaping local minima in DLS is achieved through altering the search space by utilising a cost function that depends on the weights. The utlization of weights in the cost function is used in almost all DLS algorithms (i.e. DLM, SAPS, PAWS, EWS [12], COVER [13], DDFW, and recently BalancedZ [9]) where these algorithms perform the same weighting scheme such that they increase the weights of the false clauses in the current assignment while encountering a local minima and then they reduce the weights to prevent indefinite weights growth.

This study has emerged out of studying an algorithm known as Multi level Weight Distribution (mulLWD) [14] which in turn, evolved out of Divide and Distribute Fixed Weights (DDFW). DDFW and mulLWD retain the method of dividing the clauses into satisfied and unsatisfied clauses and then distributing a fixed amount of weights among the clauses. The weights are maintained during the search by the transfer of weights from satisfied to unsatisfied clauses in both mulLWD and DDFW. Moreover, it is worth mentioning that both algorithms implicitly distribute the weights among the clauses in an adaptive manner. Hence mulLWD is a completely adaptive mechanism and requires no parameter tuning, as is the case with DDFW. A crucial difference between mulLWD and DDFW is that weights are distributed from satisfied (true) clauses to unsatisfied (false) clauses by picking a satisfied clause, as a weight donor, from multiple neighbouring levels of a clause that is currently false. Where in DDFW, only one level neighbouring picking is performed. DDFW basically was derived from Pure Additive Weighting Scheme where weights are additively increased and then periodically reduced according to a weight smoothing probability wp parameter.

Moreover, it is worth mentioning that another method older than PAWS used additive instead of multiplicative weight adjustment called DLM. The difference between PAWS and DLM is that PAWS has one parameter instead of 27 parameters (in the case of DLM it contains 27 parameters but only three need tuning) of DLM. Also, PAWS utilize random flat moves.

The mulLWD was shown to outperform DDFW, PAWS, SAPS, AdaptNovelty+ [15], BalancedZ, probSAT in a wide range of problems, and was effective in handling weights for a wide range of hard satisfiability problems.

Experiments reported in this paper indicate that mulLWD could suffer from the overhead cost caused by the exploitation of the entire 2nd level neighboring areas weights during the search steps. Among the factors that could cause such cost is that the 2nd level neighboring areas might be of large size which will cause the search to stagnate until exploring the whole area. Thus, our current investigation of weight movements in mulLWD addresses the question of whether there is an alternative method to handle the cost of exploring the 2nd level neighboring areas to further achieve gains in SAT domain. In particular, we are interested in partial multi-level weight distribution mulLWD+ scheme, that exploit weighted clauses in the entire first level neighboring areas and partially in the second level neighbors so that the cost of searching the entire 2nd level neighboring areas is prevented. The partial multi-level weight distribution approach offers the advantage of partially

exploiting weights that are indirectly connected. As a result, the high degree of deterministic weight distribution from within the same neighboring area is relaxed without compromising the benefits gained by the original approach of one level weight distribution.

In the next section we provide a general background on the evolution of clause weighting algorithms. Then, we provide further details of weights movements within the first level neighboring area (which is implemented in mulLWD and DDFW). We then introduce the mulLWD+ in more detail, and provide an experimental comparison between mulLWD+ and its predecessor mulLWD. Consequently, a significant category of problems are identified where mulLWD+ has a remarkably better performance. Then we conclude our work by recommendations for applying mulLWD+ on the domain of MAXSat.

2. CLAUSE WEIGHTING FOR SAT

DLS clause weighting heuristics for SAT follow a common method in which they start with a simple process of randomly assigning all literals l_{mn} in clause $\vee_n l_{mn}$ for all clauses of a given problem (given a boolean value for each literal ∈ 0, 1). Then the weights are equally assigned to all clauses prior to the search process. The search then starts by changing the boolean value of a literal (flipping the literal value from 0 to 1 or vice versa), if it leads to a reduction of the overall number of false clauses count of the current stage of the search. Otherwise the weights of all false clauses are increased and weights for all clauses periodically smoothed (decreased).

Basically there are two main methods on deciding on when to adjust the weights. This decision is a key factor that distinguishes the weighting heuristics from one another. Some heuristics use a multiplicative method such as SAPS, other heuristics adjust weights additively such as PAWS, DLM, DDFW, and BalancedZ.

A. Partial Multi Level Weight Distribution $(mulLWD+)$

A first level neighboring clause c_s to clause c_u is defined as: if there exists at least a literal l_x that is $\in c_s$, and $\in c_u$. Furthermore, we term l_x a same sign literal in all clauses that l_x occur in, which in turn implies that its negation literal is $-l_x$. As a result, we term any two clauses $\in \mathcal{F}$, c_i and c_j neighbors if literal l_x is $\in$ clause c_i and clause c_j where $i \neq j$. Consequently, if clause ci is false it means all literals $l_{ix} \in$ clause c_i evaluate to false (the boolean value of all literals $\in$ clause $c_i = 0$, or 1 if the literal is negated). Thus, flipping literal l_x (the boolean value of literal c_{ix}) will help clause c_{ix} and all its false neighbors. On the other hand, if clause c_n has a literal $\neg l_x$ and its current value is 0 then if we change the Boolean value of literal l_x to 1 will make literal $\neg l_x$ evaluate to 0. Assuming that literal $\neg l_x$ was the only literal in clause cn that evaluates to 1 before changing its value then clause cn will be damaged by changing the value of literal l_x from 0 to 1

Basically, if literal l_x occur in clause c_i, and clause c_i has another literal l_y which occurs in a subset of clauses S where literal $l_x \neq l_y$ then all the clause $\in S$ are a second level neighboring clauses of clause c_i. The partial multi-level weight distribution $mulLWD$+ pick the first satisfied clause that occurs in the 2nd level neighborhood area, if none were found in the first level neighboring area. Where it is not the case with $mulLWD$, where all the clauses in the 2nd level neighborhood are searched and the maximum weighted satisfied clause is chosen to be a weight donor, if no such clause exists in the first level neighbors nor in the second, a randomly satisfied clause is picked as a weight donor.

The $mulLWD$ and $mulLWD$+ heuristics both use the two uniquely implemented ideas that are built in DDFW, as in algorithm 1, and 2. Firstly, both algorithms evenly distribute a fixed amount of weights across all clauses at the beginning of the search process, escaping from the traps of local minima is performed by the weight transfer from satisfied clauses to unsatisfied (false clauses). The state-of-the-art-idea is that the weights increments and decrements are done in implicitly one step. That is, a neighboring satisfied clause will donate its weights (because weights are no longer needed since the clause

has been satisfied) to unsatisfied (false) clause. In other words, weight normalization (weight decrements of satisfied clauses) is a sub-function of the weight increment step. This idea is crucially important in the weight increment process as it waives the need of deciding at which point weight reductions should be performed. Secondly and a more original idea is the exploitation of false clauses neighboring area in order to search for weights donors. This neighboring exploitation is a key factor that distinguishes between mulLWD, mulLWD+ and DDFW. For instance, DDFW utilizes a one level neighboring area search where it looks for all the satisfied clauses that are directly connected with false clauses, where with mulLWD the two level neighboring areas are checked, so the search for a satisfied clause for the weight transfer is performed by searching all the satisfied clauses that are in the neighboring area of a false clause and their neighboring clauses. For the mulLWD+, the search for a satisfied clause to be a weight donor is similar to the one level neighboring area search in DDFW and mulLWD and significantly differ in the way it searches the 2nd level neighboring area where the second level neighboring area of a false clause is exploited partially by searching for a clause that is satisfied and has enough weight to donate, once the clause is found, the second level exploitation is stopped (as in algorithm 2 line 13).

The mulLWD search the entire first and second neighboring clauses to find a clause that is satisfied and has enough weight to donate. Our experimental results reported in Fig. 4 indicates that the cost of mulLWD 2nd level neighboring areas exploitation is very high compared to mulLWD+. This further supports our new method of partially searching the second level. Moreover, it leads to the speeding up of the overall search process as discussed in the experimental results section.

3. EXPERIMENTAL RESULTS AND ANALYSIS

Our empirical study is divided in to two stages. In stage one, we studied the weights behaviors and the occurrence of local minima for both algorithms, mulLWD and mulLWD+. Both algorithms were run on the problem sets and the number of local minima, the number of first level satisfied clause picks and the number of second level satisfied clause picks are reported. Fig 1 illustrates the total number of local minima encountered by both algorithms while searching for a satisfying assignment. The number of local minima encountered by both algorithms is almost similar except that with mulLWD+ the number was slightly less. In Fig. 2 the total number of first and second level neighbors checks while searching for a weight donor is reported. The figure indicates that the

Algorithm 1 mulLWD($\mathcal{F}$, W_{init})

1: randomly instantiate each literal in $\mathcal{F}$;
2: set the weight w_i for each clause $c_i \in \mathcal{F}$ to W_{init};
3: **while** solution is not found and not timeout **do**
4: find and return a list $\mathcal{L}$ of literals causing the greatest reduction in weighted cost Δw when flipped;
5: **if** ($\Delta w < 0$) or ($\Delta w = 0$ and probability $\leq 15\%$) **then**
6: randomly flip a literal in $\mathcal{L}$;
7: **else**
8: **for** each false clause c_f **do**
9: search for a satisfied same sign neighbouring clause c_k with maximum weight w_k;
10: **if** C_K found **then**
11: transfer a weight of one from c_k to c_f;
12: **else**
13: select a satisfied same sign neighbour of neighbouring clause c_k with maximum weight w_k;
14: **end if**
15: **if** $w_k < Init$ **then**
16: randomly select a clause c_k with weight $w_k \geq W_{init}$;
17: **end if**
18: **if** $w_k >= W_{init}$ **then**
19: transfer a weight of one from c_k to c_f;
20: **end if**
21: **end for**
22: **end if**
23: **end while**

Algorithm 2 mulLWD+($\mathcal{F}$, W_{init})

```
randomly instantiate each literal in F;
set the weight w_i for each clause c_i ∈ F to W_init;
while solution is not found and not timeout do
    find and return a list L of literals causing the greatest reduction in
    weighted cost Δw when flipped;
    if (Δw < 0) or (Δw = 0 and probability ≤ 15%) then
        randomly flip a literal in L;
    else
        for each false clause c_f do
            search for a satisfied same sign neighbouring clause c_k with
            maximum weight w_k;
            if C_K found then
                transfer a weight of one from c_k to c_f;
            else
                select the first satisfied same sign neighbour of neighbour-
                ing clause c_k;
            end if
            if w_k < Init then
                randomly select a clause c_k with weight w_k ≥ W_init;
            end if
            if w_k >= W_init then
                transfer a weight of one from c_k to c_f;
            end if
        end for
    end if
end while
```

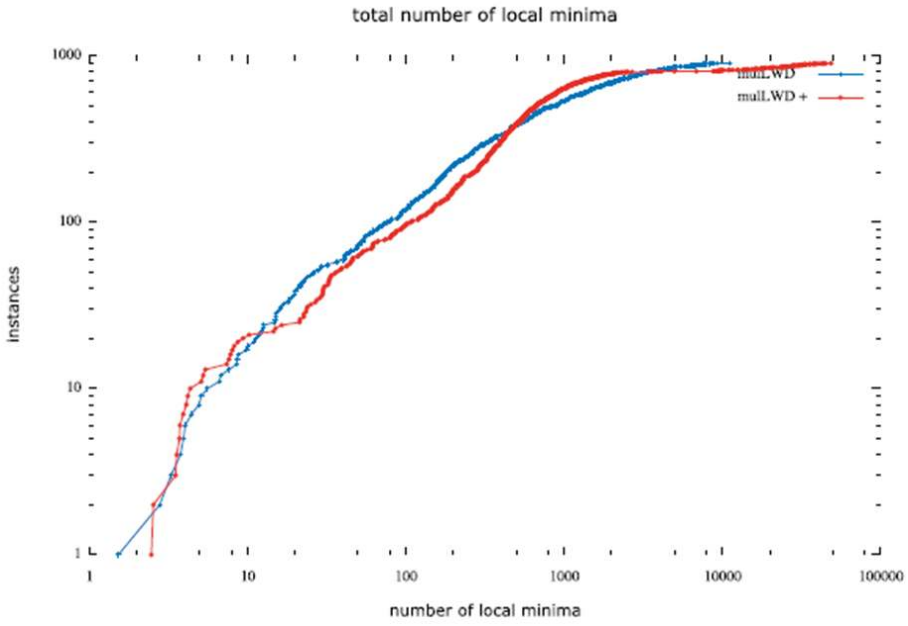

Fig. 1. Illustration of the total number of local minima encountered by mulLWD and mulLWD+

total number of checks for both algorithms was also similar. We also reported the number of first level satisfied clause picks for both algorithms as in Fig. 3. The behaviors of the algorithms differ significantly as mulLWD+ successfully picks a first level neighboring clause much more than mulLWD, where as in the second level neighboring clause picks, mulLWD spend most of the time, as in Fig. 4.

The behaviors of mulLWD and mulLWD+ as shown in the mentioned figures indicates that the mulLWD+ partial by second level neighboring area weight distribution enhances the overall

search process while retaining escaping from traps caused by local minima. That is shown in Figs. (1, 2, 3, 4) where the number of the total checks, number of local minima encountered, remained the same while algorithm mulLWD+ spent much less time searching the 2nd level neighboring area and concentrated on the first level neighbors.

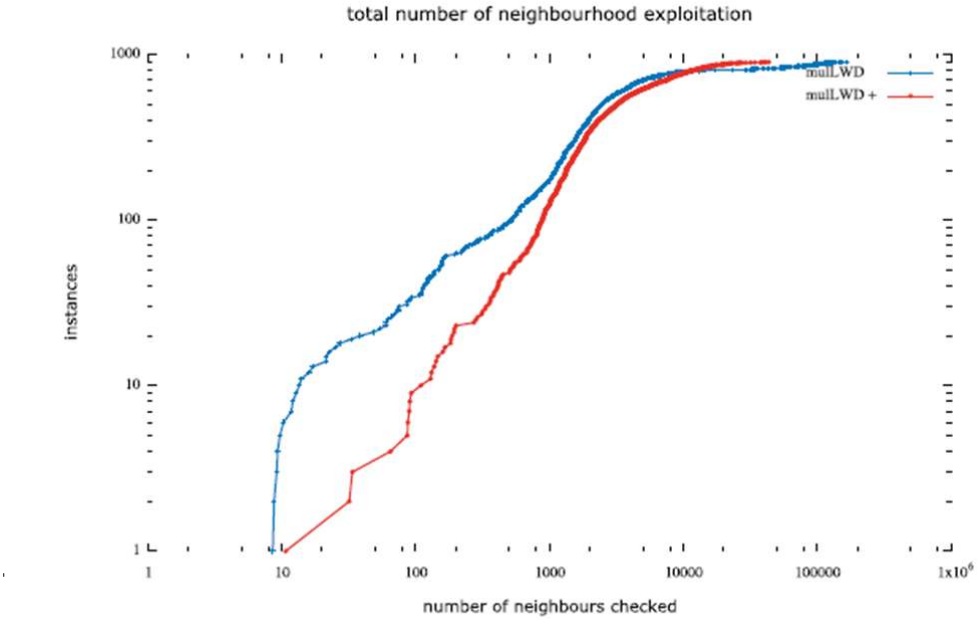

Fig. 2. Illustration of the number of neighboring clauses checks in both mulLWD and mulLWD+

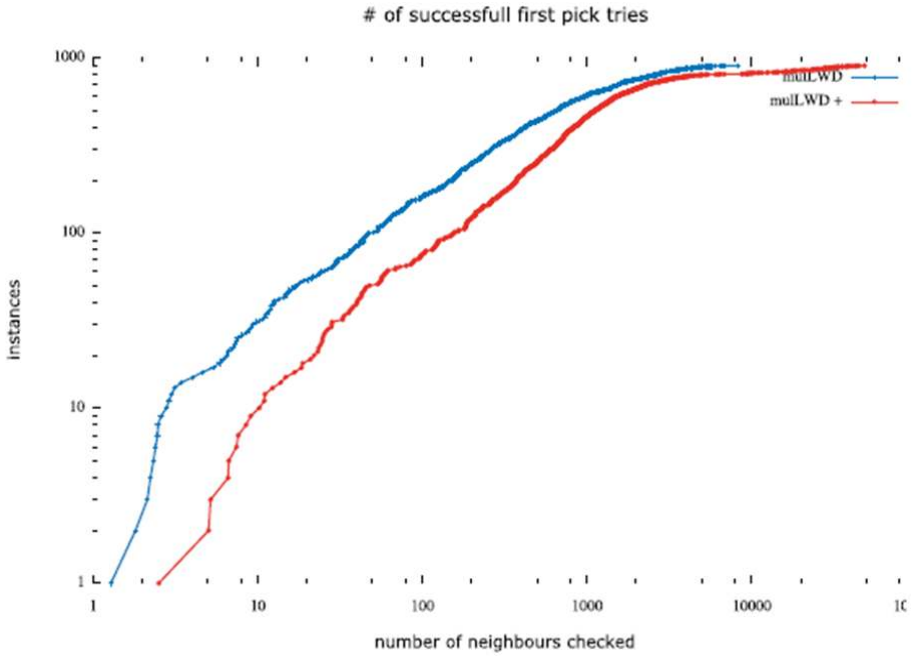

Fig. 3. Illustration of the total number of first level neighboring clause pick in both mulLWD and mulLWD+

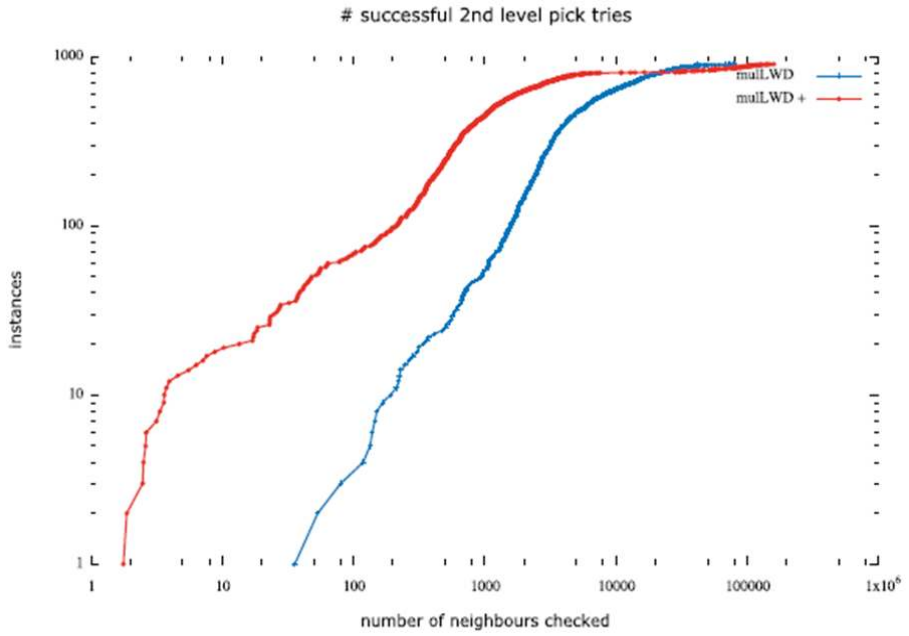

Fig. 4. Illustration of the total number of 2nd -neighbor level clause pick in both mulLWD and mulLWD+

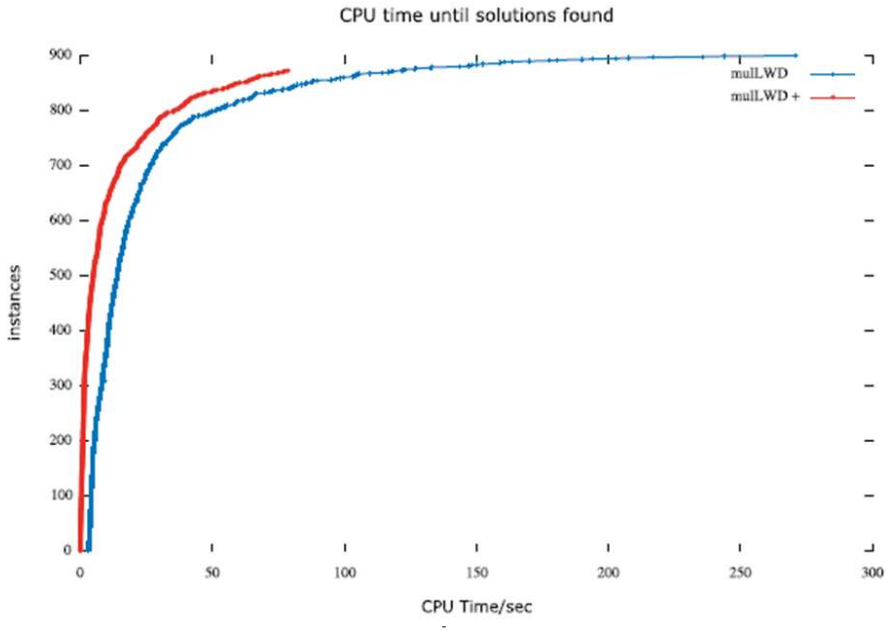

Fig. 5. Comparison between mulLWD and mulLWD+, the results are recorded in seconds

In stage two, we compared the performance of mulLWD and mulLWD+. The comparison experiments were performed on a iMAC machine with i5 multiCore 2.5. GHz CPU and 8GB memory. The experiment general sittings were set as follows for both algorithms: the cutoff was set 50,000,000 for each run, on all the problem set. The algorithms were allowed 1000 tries on each problem. For each run the time and % of solution, if found, is reported.

Table 1. Comparison results of MULLWD, and MULLWD+. for each algorithm the time and the % solved is reported

Problem	mulLWD+		mulLWD	
	Time	%solved	Time	%solved
ais	22.77	100	41.32	100
bw.c	535.52	100	1399.96	94
bw.d	1231.78	100	4539.66	87
f800-hard	170	100	597.03	100
f1600-hard	324.33	100	1181.18	92
flat200	159.56	100	422.01	100
g125.18	150.49	100	312.49	100
logistics	15.91	100	57.08	100
uf400-hard	64.48	100	266.36	100
Unif-k6-r4.37v135	2517	100	5913	100

Fig. 5 illustrates the CPU time in which the algorithms could reach solutions in all attempts. As discussed above, the narrowing of neighboring areas pays off as mulLWD+ could outperform mulLWD in all runs. Furthermore, mulLWD+ reached a 100% successful rate in every attempt, where mulLWD could only reach a success rate of 94% on the blocks world (bw.c) problem, and 87% on the (bw.d) and 92% on (f1600) problem, and a 100% success rate on the remaining problem set. Furthermore, mulLWD+ was faster in reaching solutions by almost a factor of four as reported in Table 1.

4. CONCLUSION

Overall we can conclude that the addition of a the partial multi-level weight distribution mulLW-D+ has shown a significant enhancement over the 2nd level neighboring areas exploitation over the entire range of the problem sets we have considered. Thus, the reduction of the 2nd level neighboring areas exploitation clearly enhances the overall process of searching for solutions while not compromising the efficiency of the search process. Thus due to the fact that MulLWD+ retains the first level neighborhood exploitation and spends much less in the exploitation of the 2nd level neighboring areas which incurs a high overhead cost.

ACKNOWLEDGEMENT

The authors would like to acknowledge the financial support of the Scientific Research Committee at University of Petra. Also we would like to thank all faculty members of the Information Technology faculty who contributed directly or indirectly to this work.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Holger Hoos, Thomas Stulze. Stochastic local search. Morgan Kaufmann, Cambridge, Massachusetts; 2005.
2. Morris P. The breakout method for escaping from local minima. In Proceedings of 11th AAAI. 1993;40–45.
3. Byungki Cha, Kazuo Iwama. Adding new clauses for faster local search. In Proceedings of 13th AAAI. 1996;332–337.
4. Jeremy Frank. Learning short-term clause weights for GSAT. In Proceedings of 15th IJCAI. 1997;384–389.
5. Wu Z, Benjamin Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In Proceedings of 17th AAAI. 2000;310–315.
6. Frank Hutter, Dave Tompkins, Holger Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In Proceedings of 8th CP. 2002;233–248.
7. John Thornton, Pham DN, Stuart Bain, Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In Proceedings of 19th AAAI. 2004;191–196.
8. Abdelraouf Ishtaiwi, John Thornton, Abdul Sattar, Duc Nghia Pham. Neighbourhood clause weight redistribution in local search for SAT. In Proceedings of 11th CP. 2005;772–776.
9. Chumin Li, Chong Huang, Ruchu Xu. Balance between intensification and

diversification: Two sides of the same coin. In Proc. of SAT Competition, Solver Description. 2013;10–11.
10. Adrian Balint, Armin Biere, Andreas Fro¨ hlich, Uwe Scho¨ ning. Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses. Springer International Publishing, Cham. 2014;302–316.
11. Chuan Luo, Shaowei Cai, Wei Wu, Kaile Su. Double configuration checking in stochastic local search for satisfiability. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI'14. 2014;2703–2709.
12. Shaowei Cai, Kaile Su, Qingliang Chen. EWLS: A new local search for minimum vertex cover. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI, Atlanta, Georgia, USA; 2010.
13. Silvia Richter, Malte Helmert, Charles Gretton. A stochastic local search approach to vertex cover. In KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabru¨ ck, Germany. Proceedings. 2007;412–426.
14. Abdelraouf Ishtaiwi, Marco Juarez, Ghassan Issa. Multi level weight distribution in dynamic local search for SAT. In Proceedings of 3rd International Conference on Information Technology, Control and Computer Engineering ITCCE. 2017;17:79–86.
15. Holger Hoos. An adaptive noise mechanism for Walk SAT. In Proceedings of 19th AAAI. 2002;655–660.

Peer-review history:
The peer review history for this paper can be accessed here:
http://sciencedomain.org/review-history/22549